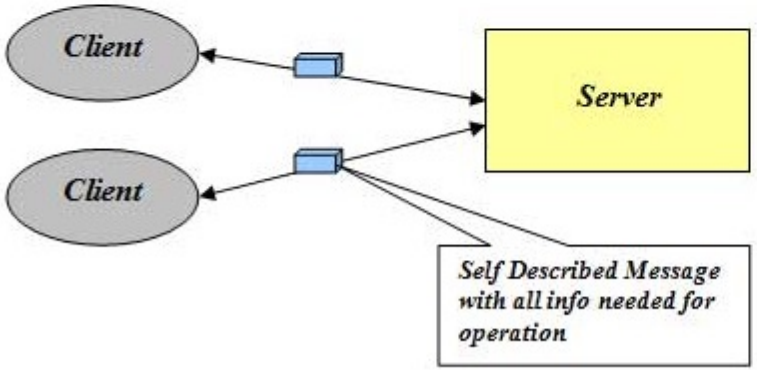
E6156 – Topics in SW Engineering: Cloud Computing

Lecture 8: Cool Stuff

Stateless, Sessions Example

Stateless

The notion of statelessness is defined from the perspective of the server. The constraint says that the server should not remember the state of the application. As a consequence, the client should send all information necessary for execution along with each request, because the server cannot reuse information from previous requests as it didn't memorize them. All info needed is in message.



Statelessness

2. Application State vs Resource State

It is important to understand the between the application state and the resource state. Both are completely different things.

Application state is server-side data that servers store to identify incoming client requests, their previous interaction details, and current context information.

Resource state is the current state of a resource on a server at any point in time – and it has nothing to do with the interaction between client and server. It is what we get as a response from the server as the API response. We refer to it as resource representation.

> REST statelessness means being free from the application state.

3. Advantages of Stateless APIs

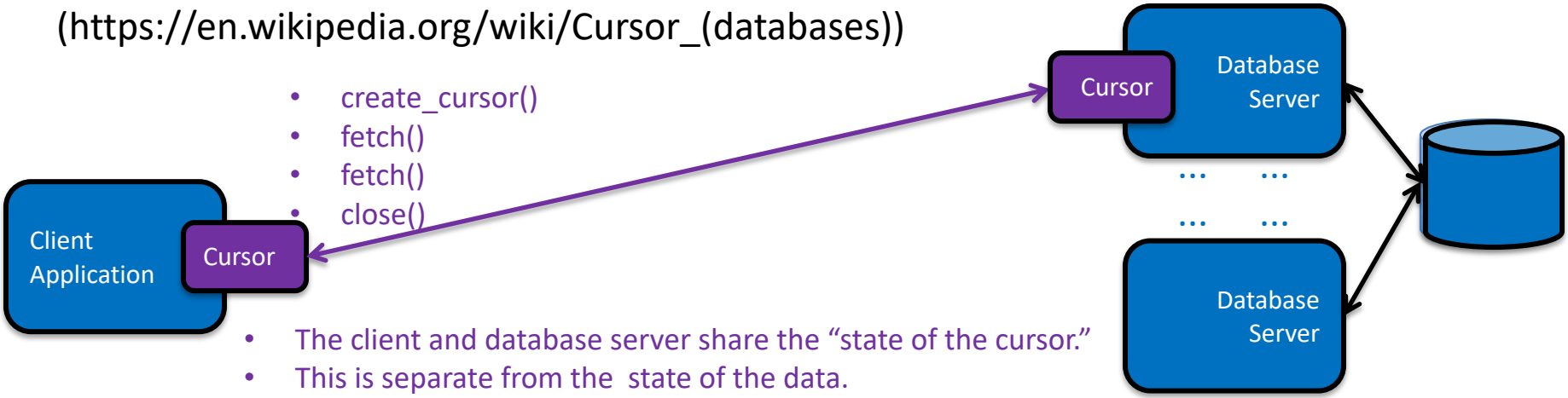
There are some very noticeable advantages of having **REST APIs stateless**.

1. Statelessness helps in **scaling the APIs to millions of concurrent users** by deploying it to multiple servers. Any server can handle any request because there is no session related dependency.
2. Being stateless makes REST APIs **less complex** – by removing all server-side state synchronization logic.
3. A stateless API is also **easy to cache** as well. Specific softwares can decide whether or not to cache the result of an HTTP request just by looking at that one request. There's no nagging uncertainty that state from a previous request might affect the cacheability of this one. It **improves the performance** of applications.
4. The server never loses track of "where" each client is in the application because the client sends all necessary information with each request.

- This concept can be confusing with experience with other approach to session state.
- Some application frameworks did/do keep session state on the server.
- Application servers and frameworks have various support for "session state," e.g. https://flask-session.readthedocs.io/en/latest/

Simple Example from Databases

- "In computer science, a database cursor is a mechanism that enables traversal over the records in a database. Cursors facilitate subsequent processing in conjunction with the traversal, such as retrieval, addition and removal of database records. The database cursor characteristic of traversal makes cursors akin to the programming language concept of iterator."
(https://en.wikipedia.org/wiki/Cursor_(databases))



- The client and database server share the "state of the cursor."
- This is separate from the state of the data.
- This hinders scalability and fault tolerance if there are multiple database servers in a cluster.

Some Code

```python
def t_cursor():

    print("\nUsing cursors ...")
    sql = "select * from users";

    for i in range(0,5):
        if i == 0:
            res = with_cursor(done=False, sql=sql)
        elif i == 4:
            res = with_cursor(done=True)
        else:
            res = with_cursor()

        print("Res = ", res)
```

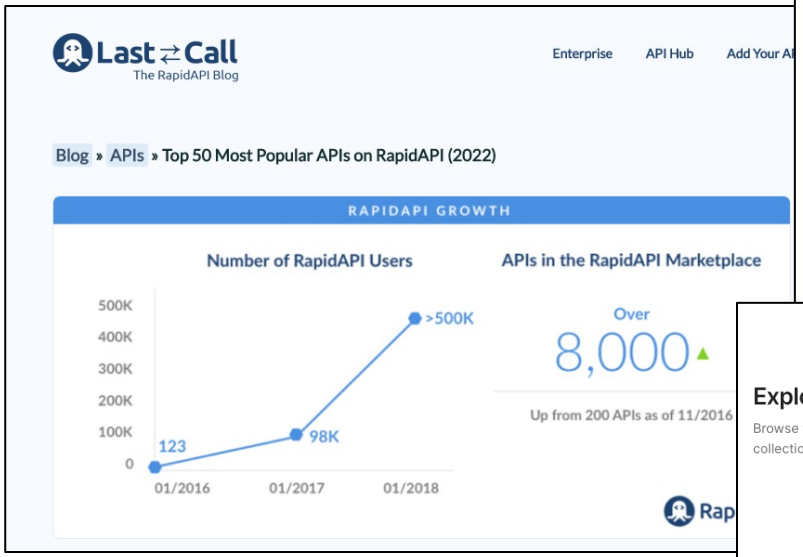
```python
def t_cursor():

    print("\nUsing cursors ...")
    sql = "select * from users";

    for i in range(0,5):
        if i == 0:
            res = with_cursor(done=False, sql=sql)
        elif i == 4:
            res = with_cursor(done=True)
        else:
            res = with_cursor()

        print("Res = ", res)
```

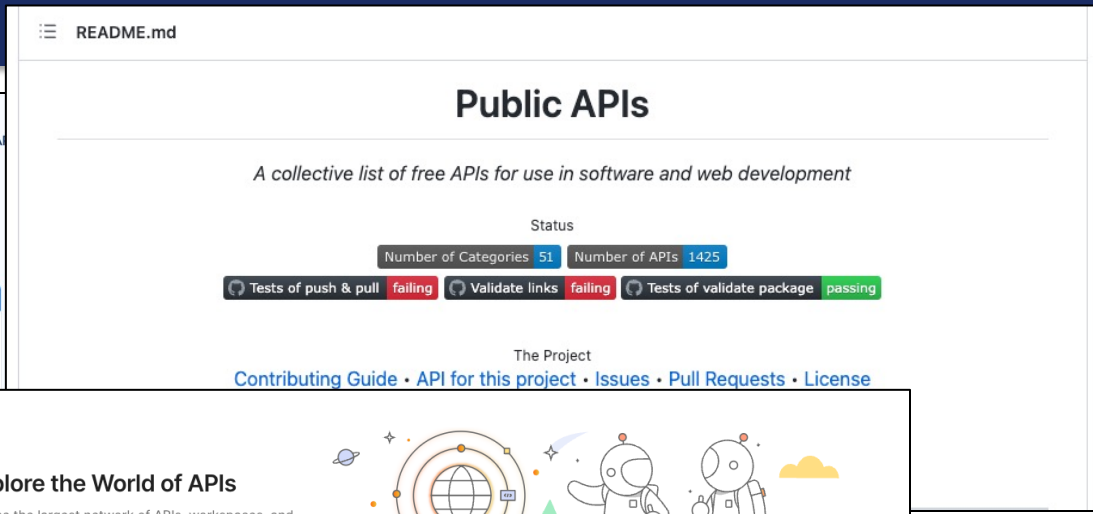
```python
def t_without_cursor():

    print("\nNot using cursors.")

    sql = "select * from users";

    for i in range(0,5):
        res = without_cursor(sql=sql, offset=i)
        print("Res = ", res)
```

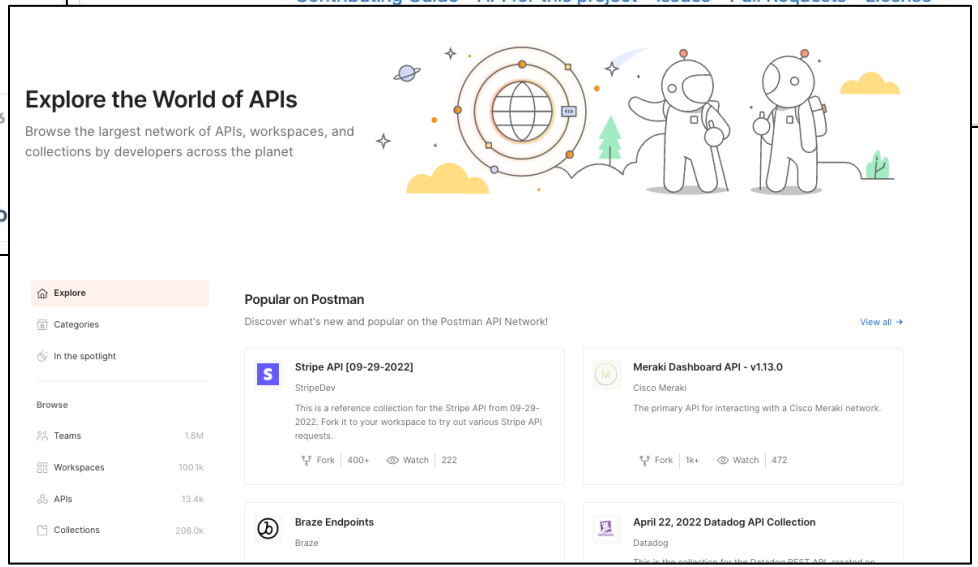
```python
def without_cursor(sql, offset):

    cursor = conn.cursor()
    tmp_sql = sql + " limit 1" + " offset " + str(offset)
    res = cursor.execute(tmp_sql)
    res = cursor.fetchone()
    cursor.close()
    return res
```

Calling APIs

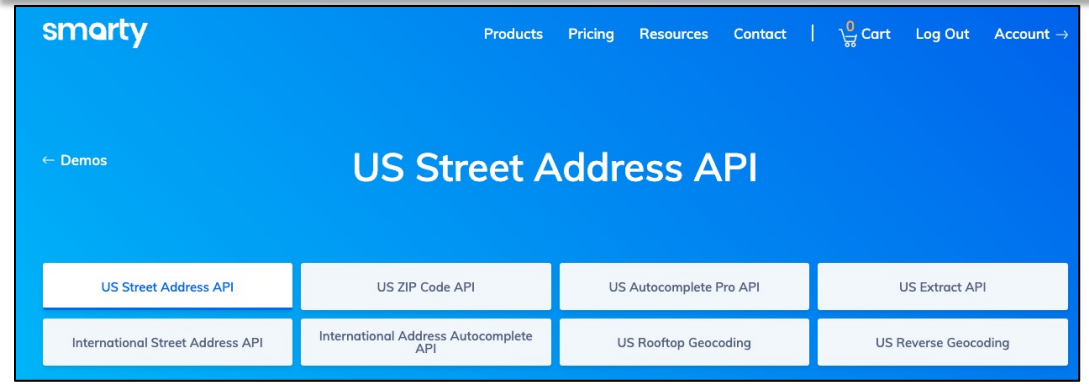
A World of APIs

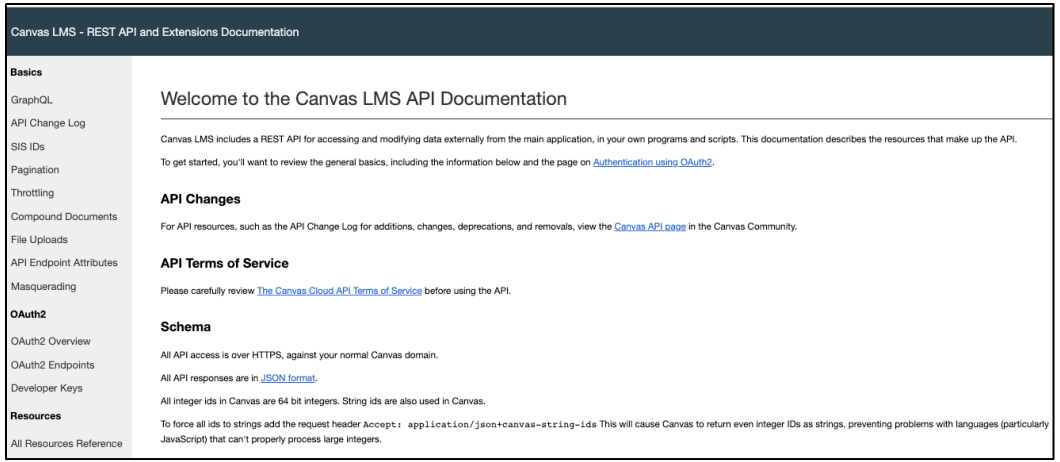






Two APIs I Use in My Project

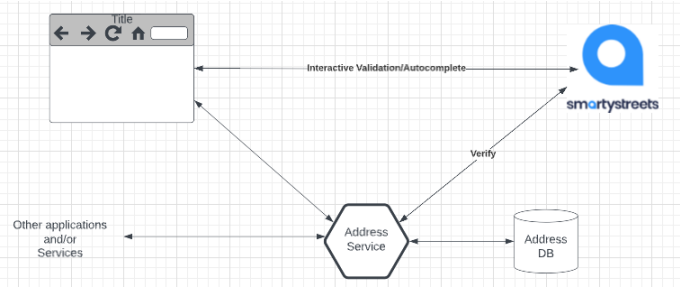


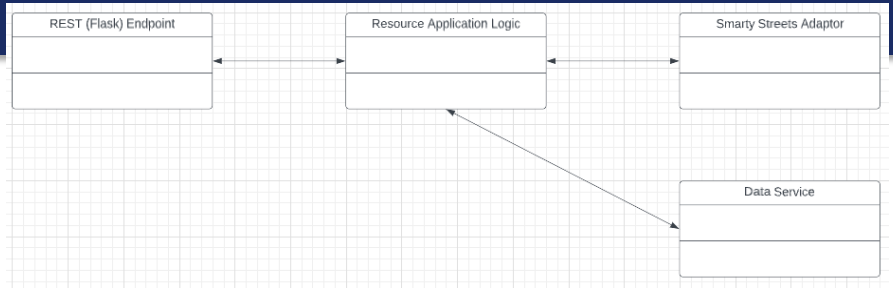


- Addresses:
  - People are really bad at entering mailing addresses.
    - 520 w 120th st NYC
    - 520 West 120 St New York, NY
    - … …
  - We want a canonical, verified representation.

- CourseWorks
  - Students will access my web application to set up projects, teams, …
  - I want to verify that students are in the class, and keep list of students up to date.

Address Example

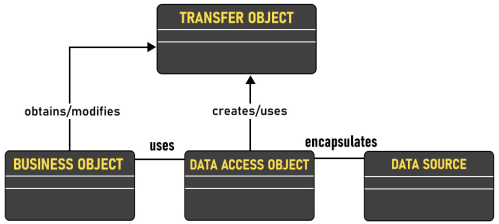




- Use autocomplete functions to help users correctly enter addresses https://jsfiddle.net/smartystreets/j1Lq5say
- But, … …
  - A microservice cannot assume that a valid UI is calling the service.
  - It must support any UI or endpoint authorized to call the service.
  - So, we will also validate in the microservice implementation.
  - Example code walkthrough and online tools.

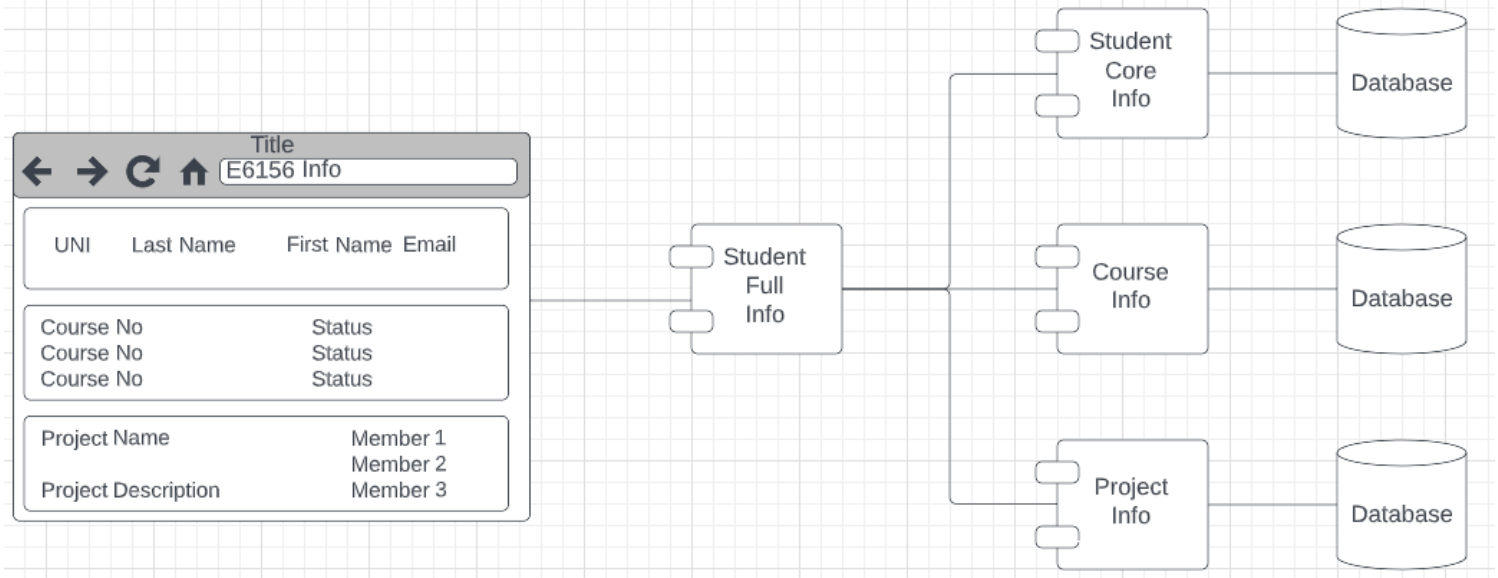
Microservice Implementation

1. Receive JSON body in PUT/POST.
2. Call API to verify.
3. If verified, save in DB.
4. URL path is the delivery point
   1. Zipcode
   2. Zip plus 4
   3. Delivery point



Synchronous versus Asynchronous

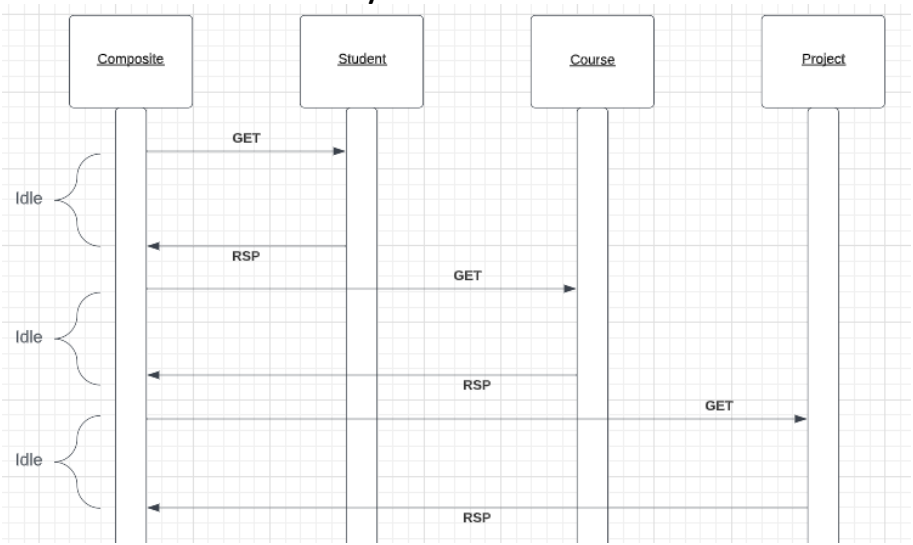
Synchronous versus Asynchronous



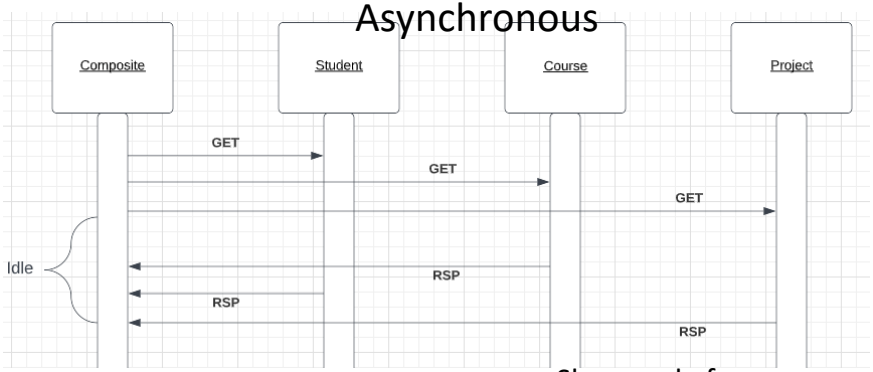
- CRUD on composite service StudentFullInfo requires CRUD on 3 microservices.
- There are two styles for calling the component (composed services)
  - Synchronous, one after another, waiting each time.
  - Asynchronous: Call all three in parallel, and ”gather the answer.”

Synchronous versus Asynchronous

Synchronous



Asynchronous



Show code from:

- call_urls_synch_asynch
- async3
- async_p_mysql

Async-python

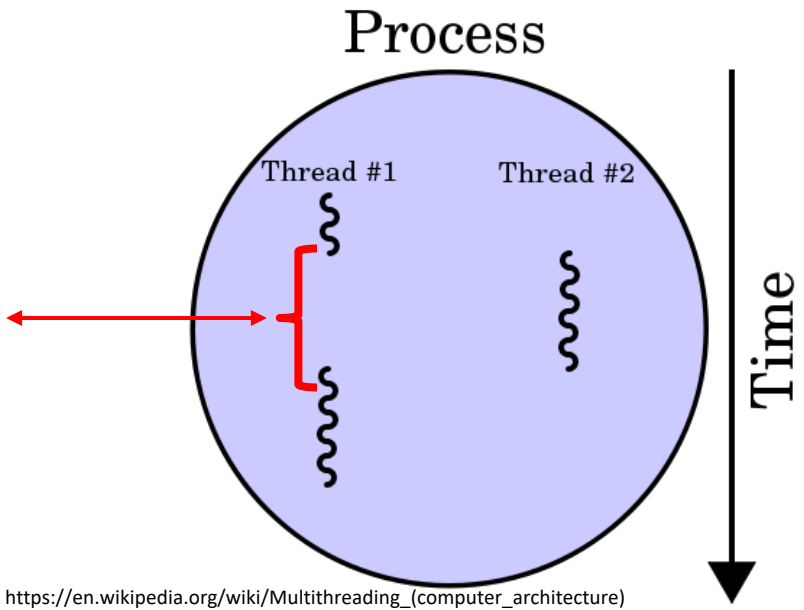
- av_requests
- av_async_run

- The elapsed time for calling synchronous is *the sum of the calls' response times.*
- The elapsed time for calling asynchronous is the maximum response time of the 3 calls.
- Asynchronous:
  - Has better elapsed times and less "application is idle waiting" time, but
  - The logic is more complex, especially if the composite services has to "undo" some calls because others failed.

Concurrency

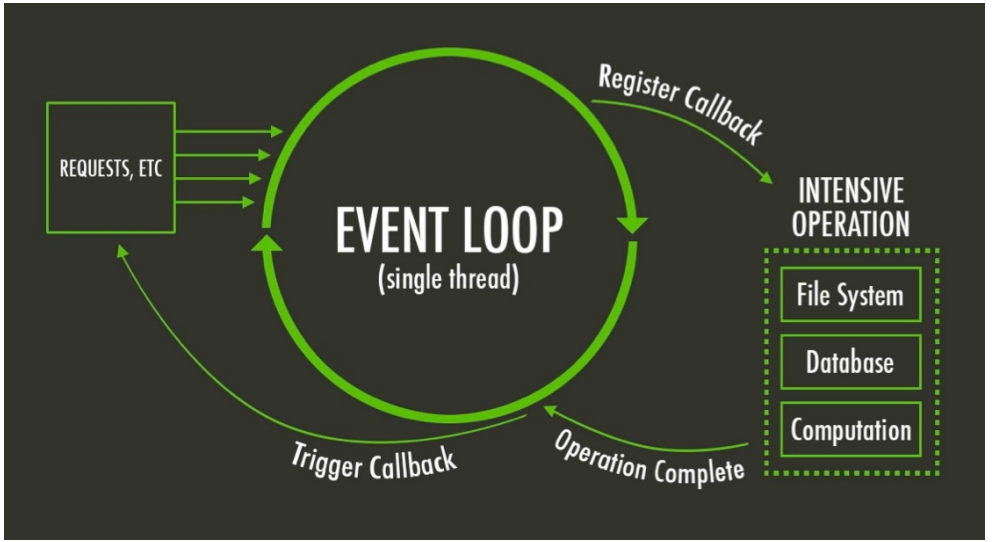
- A running program executes in an operating system *process.* There are many approaches to concurrency:
  - Multiprocessing:
    - The system has several “active processes.”
    - Only one process is executing on a CPU (core) as a time.
    - The OS switches between processes when the running process performs a blocking operations, e.g. file IO, remote network call, ...
  - Multithreading:
    - There are several “active threads” within a process.
    - (Usually) Only one thread is executing on a CPU (core) at a time.
    - When the thread performs a blocking operation, the process continues to run but executes a different, ready thread.
  - Coroutines:
    - There is a single active thread within a process.
    - When a thread performs a blocking operation, the programming language runtime does weird internal stuff
      - Makes a copy of the current coroutines environment (call stack, ...)
      - Starts a ready coroutine by switching to its call stack and next statement.
    - The thread and process do not block.
- A *context switch* is changing the currently running process, thread or task.
  - There is a performance overhead for a context switch.
  - Overhead(multiprocessing) >> Overhead(multithreading) >> Overhead(coroutines)

Threads, Promises, Asynch Functions, ...



https://en.wikipedia.org/wiki/Multithreading_(computer_architecture)

- Multithreading
  - Thread #1 “blocks” for an IO or API call.
  - Process switches to “ready” thread #2



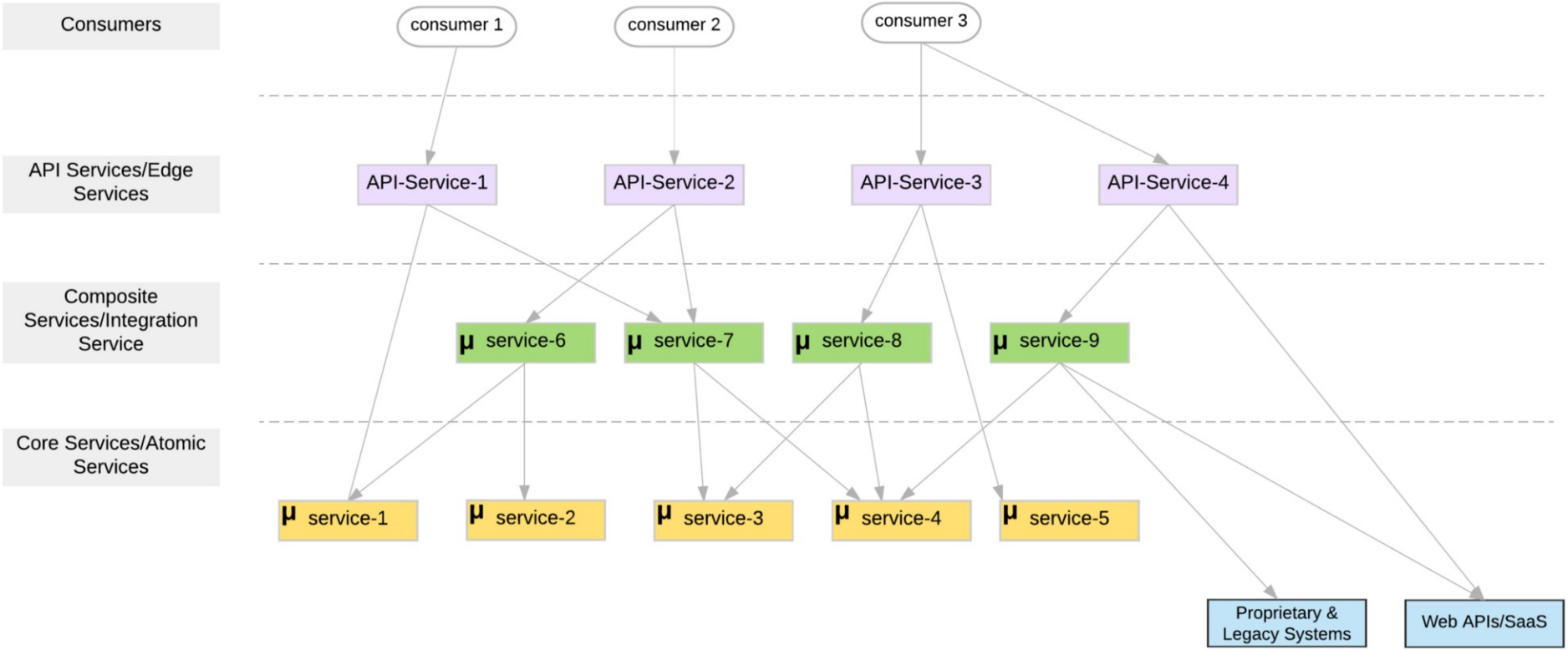
- Callbacks

https://medium.com/@amragaey/shortly-how-node-js-works-on-a-single-thread-763fda99f012

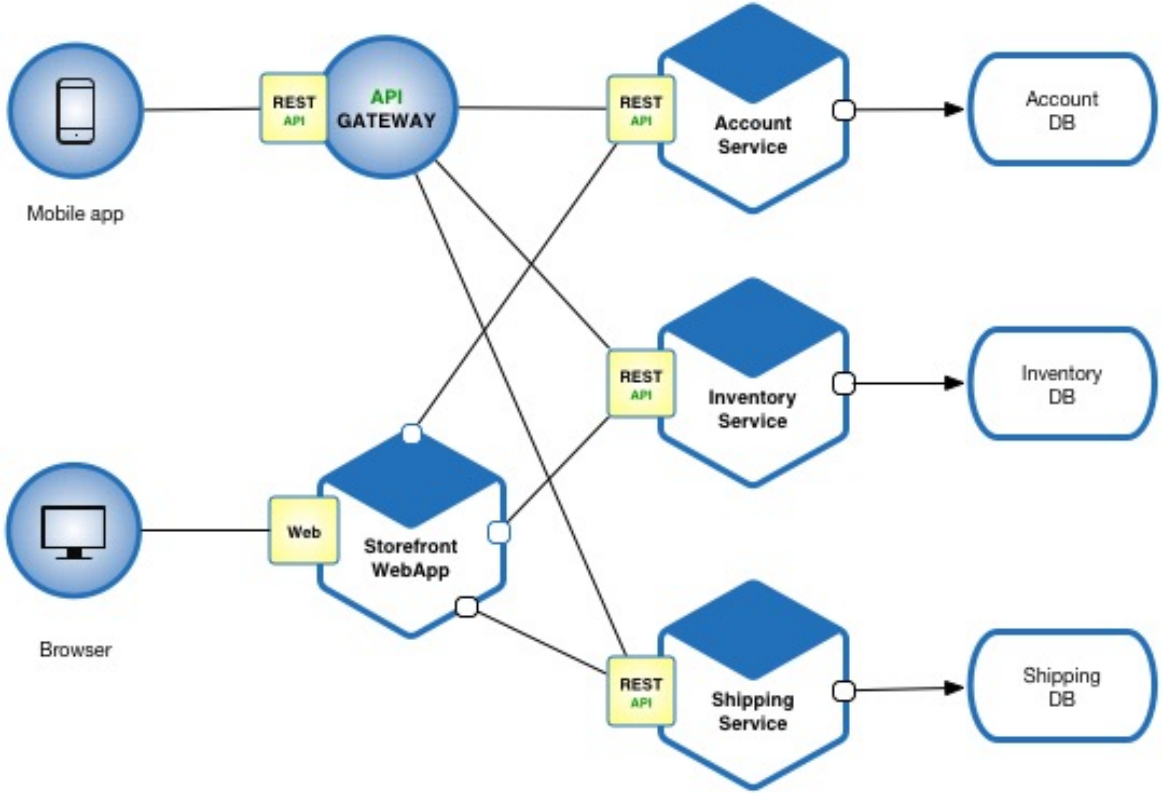
  - Incoming requests map to an event.
  - There is a single thread running application code associated with an event..
  - Long running calls get assigned to a separate worker thread, and register a callback.
  - Completion of an operation results in an event, which resumes the application via a callback.

Service Composition

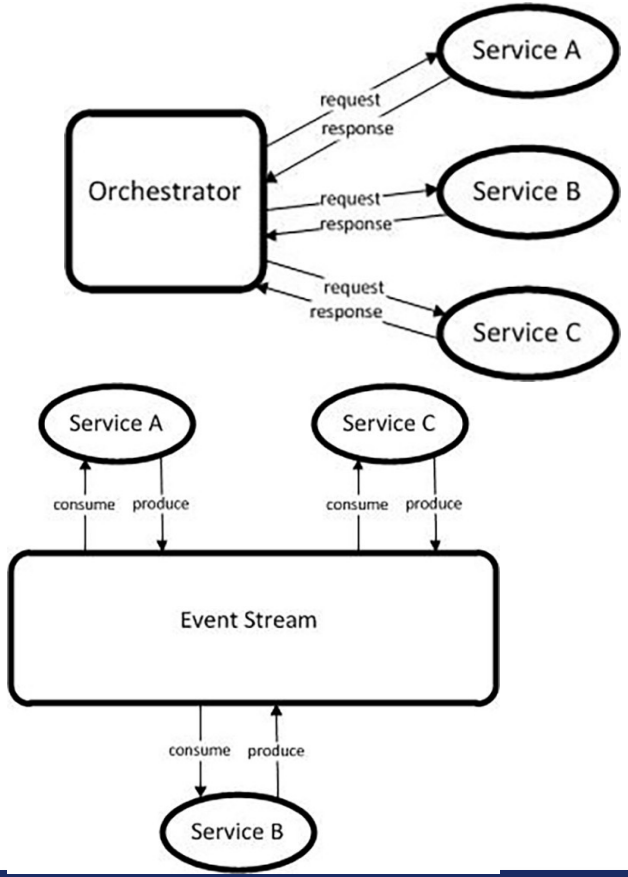
Microservice Layers

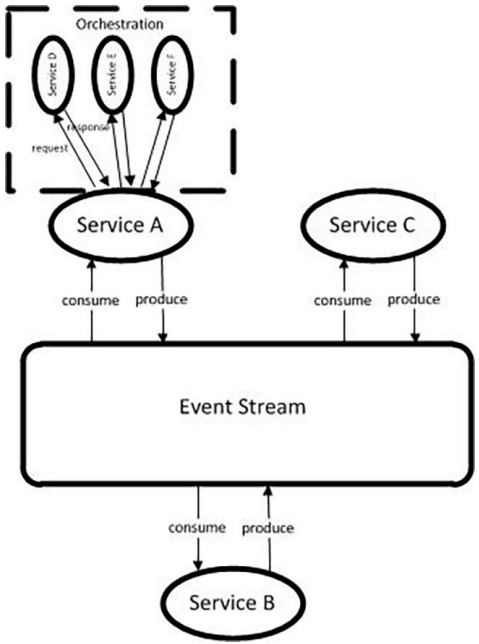


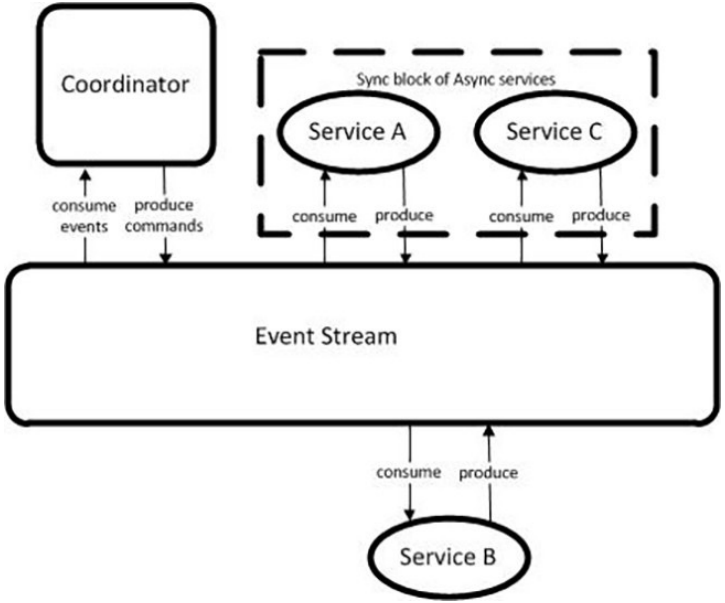
Another View



Models (https://medium.com/capital-one-tech/microservices-when-to-react-vs-orchestrate-c6b18308a14c)





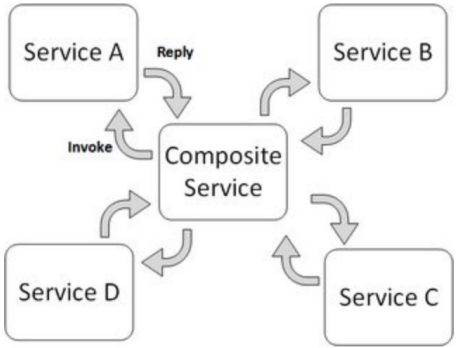


Concepts

Service orchestration

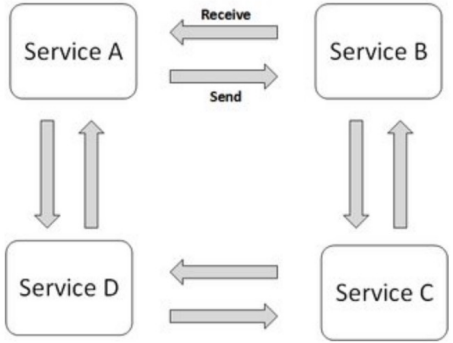
Service orchestration represents a single centralized executable business process (the orchestrator) that coordinates the interaction among different services. The orchestrator is responsible for invoking and combining the services.

The relationship between all the participating services are described by a single endpoint (i.e the composite service). The orchestration includes the management of transactions betweer individual services. Orchestration employs a centralized approach for service composition.



Service Choreography

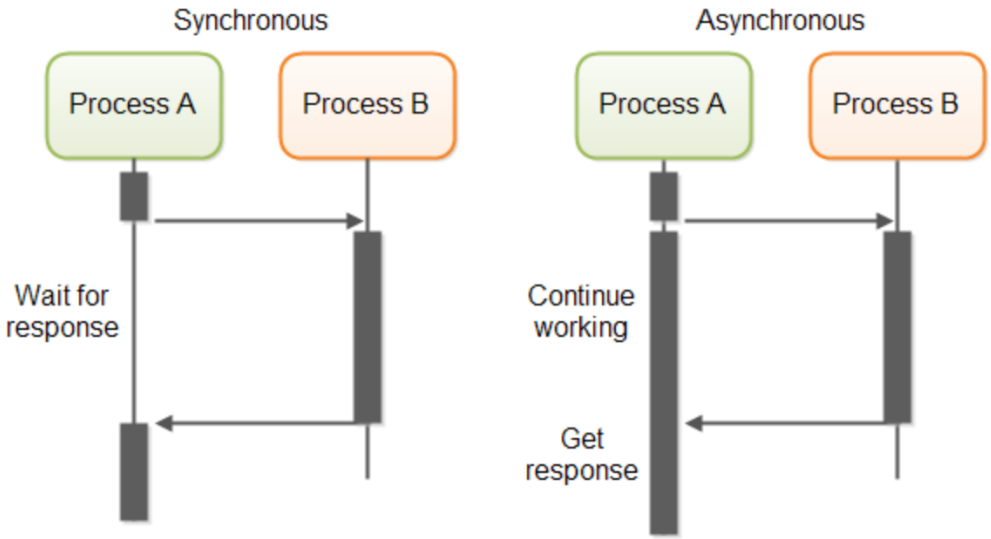
Service choreography is a global description of the participating services, which is defined by exchange of messages, rules of interaction and agreements between two or more endpoints. Choreography employs a decentralized approach for service composition.



The choreography describes the interactions between multiple services, where as orchestration represents control from one party's perspective. This means that a **choreography** *differs* from an **orchestration** with respect to where the logic that controls the interactions between the services involved should reside.

These are not formally, rigorously defined terms.

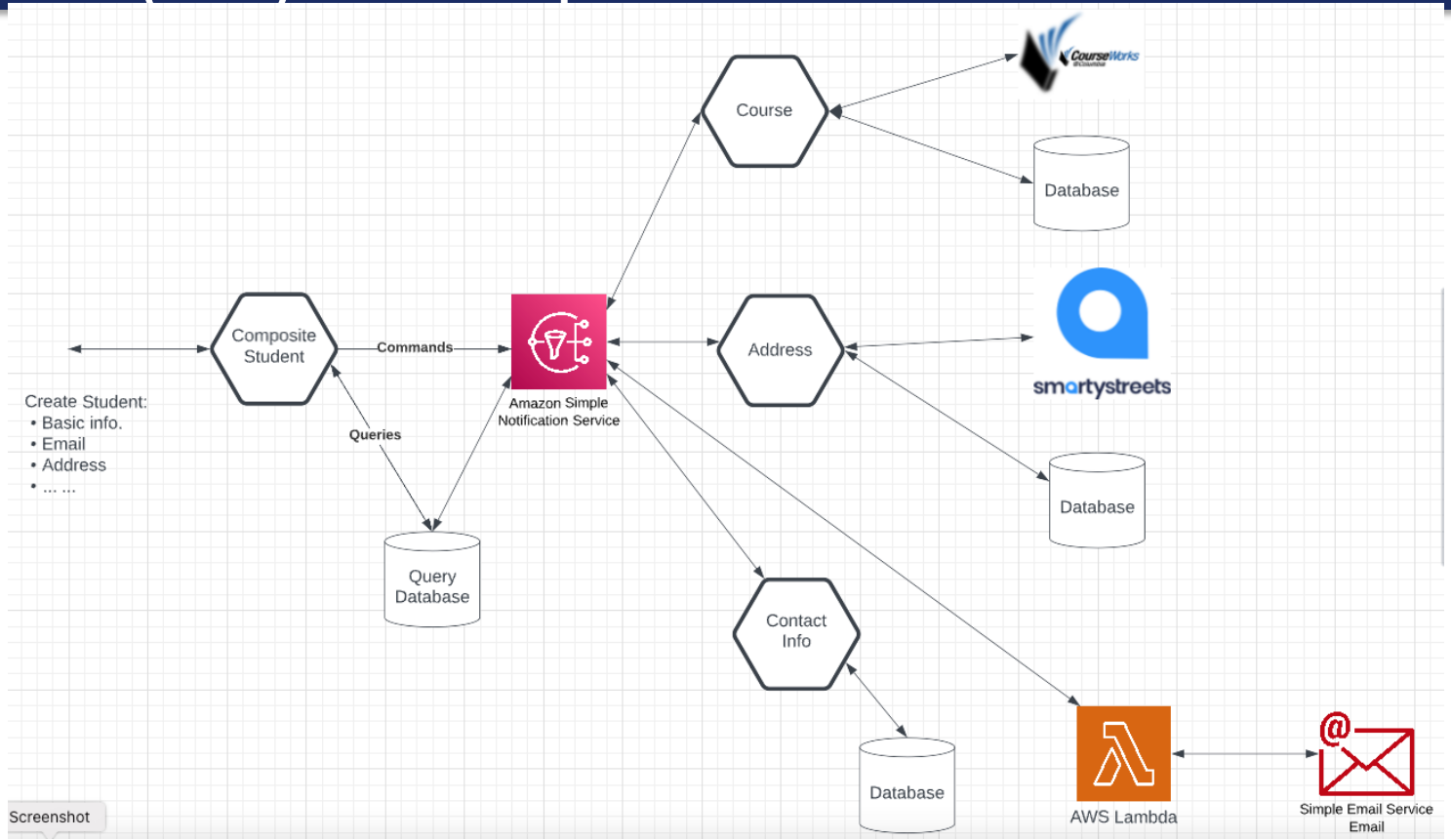
Service Orchestration/Composition



RESTFul HTTP calls can be implemented in both a synchronous and asynchronous fashion at an IO level.

- One call to a composite service
  - May drive calls to multiple other services.
  - Which in turn, may drive multiple calls to other services.
- Synchronous (Wait) and calling one at a time, in order is
  - Inefficient
  - Fragile
  - … …
- Asynchronous has benefits but can result in complex code.

What I (We) Will Implement



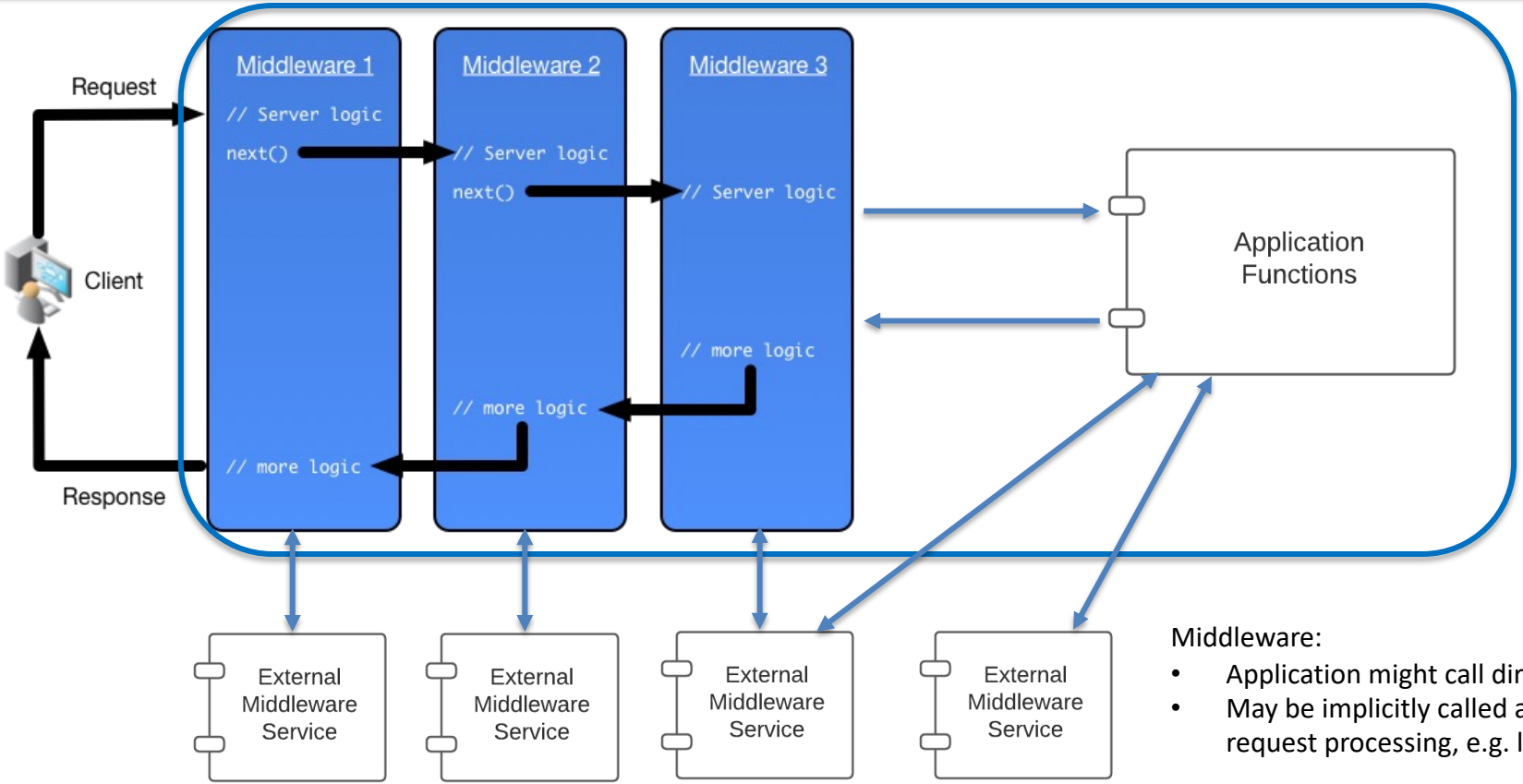
Middleware

What is middleware?

Middleware is software that lies between an operating system and the applications running on it. Essentially functioning as hidden translation layer, middleware enables communication and data management for distributed applications. It's sometimes called plumbing, as it connects two applications together so data and databases can be easily passed between the "pipe." Using middleware allows users to perform such requests as submitting forms on a web browser, or allowing the web server to return dynamic web pages based on a user's profile.

Common middleware examples include database middleware, application server middleware, message-oriented middleware, web middleware, and transaction-processing monitors. Each program typically provides messaging services so that different applications can communicate using messaging frameworks like simple object access protocol (SOAP), web services, representational state transfer (REST), and JavaScript object notation (JSON). While all middleware performs communication functions, the type a company chooses to use will depend on what service is being used and what type of information needs to be communicated. This can include security authentication, transaction management, message queues, applications servers, web servers, and directories. Middleware can also be used for distributed processing with actions occurring in real time rather than sending data back and forth.

Middleware – Conceptual Model



Middleware:

- Application might call directly, e.g. DB
- May be implicitly called as part of request processing, e.g. logging.

Middleware and Flask

```python
# These methods get called before/after. You can check security information for all requests.
@application.before_request
def before_decorator():
    a_ok = sec.check_authetication(request)
    print("a_ok = ", a_ok)
    if a_ok[0] != 200:
        handle_error(a_ok[0], a_ok[1], a_ok[2])


@application.after_request
def after_decorator(rsp):
    print("... In after decorator ...")
    notify.notify(request, rsp)
    return rsp
```

- White list of URLs
- If not on white list:
  - Verify token
  - Verify authorization

- List of filters of the form:
  - URL
  - HTTP Method
  - SNS topic to use
- Send an event to SNS if matches filter

```python
@application.route("/api/users", methods=["GET", "POST"])
def users_get():...
```

Show some demo code.

So, How Will We Use This?

- Add before_request and after_request to microservices. (Note: There are other approaches).
- Implement two middleware interceptors:
  - Security
    - Configured with a JSON file.
    - File lists which {path, method} not being logged on.
    - While IDs can perform which methods on which paths.
    - Automates driving login redirect (401 – Unauthorized)
    - Checks authorization (403– Forbidden)
  - Notification:
    - Configured with JSON in environment variables.
    - File lists which operations emit a before and/or after event to an SNS topic.
- We will use notification to drive some other things like webhooks.

- I will demo middleware and webhook today.
- Your project will use middleware and SNS.

Cloud Identity

(Open ID, OAuth2, ...)

Reminder

/Users/donaldferguson/Dropbox/00NewProjects/google_login

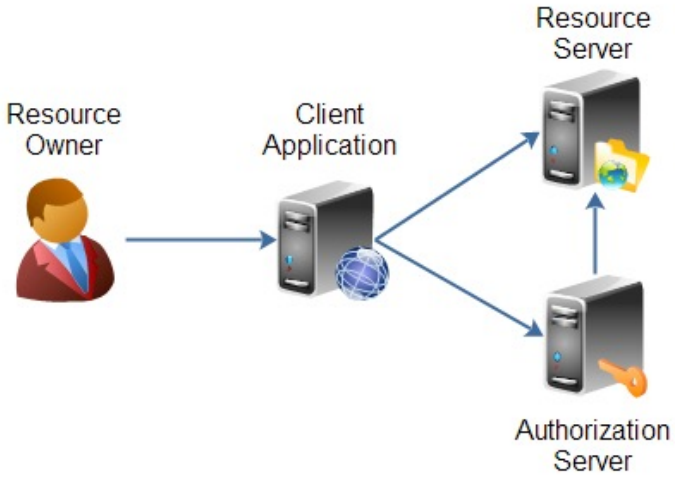
Overview (http://tutorials.jenkov.com/oauth2/index.html)

- Resource Owner
  - Controls *access* to the "data."
  - Facebook **user**, LinkedIn **user**, …
- Resource Server
  - The website that holds/manages info.
  - Facebook, LinkedIn, ...
  - And provides access API.
- Client Application
  - "The product you implemented."
  - Wants to read/update
    - "On your behalf"
    - The data the data that the Resource Server maintains
- Authorization Server
  - Grants/rejects authorization
  - Based on Resource Owner decisions.
  - Usually (logically) the same as Resource Server.

OAuth 2.0 defines the following roles of users and applications:

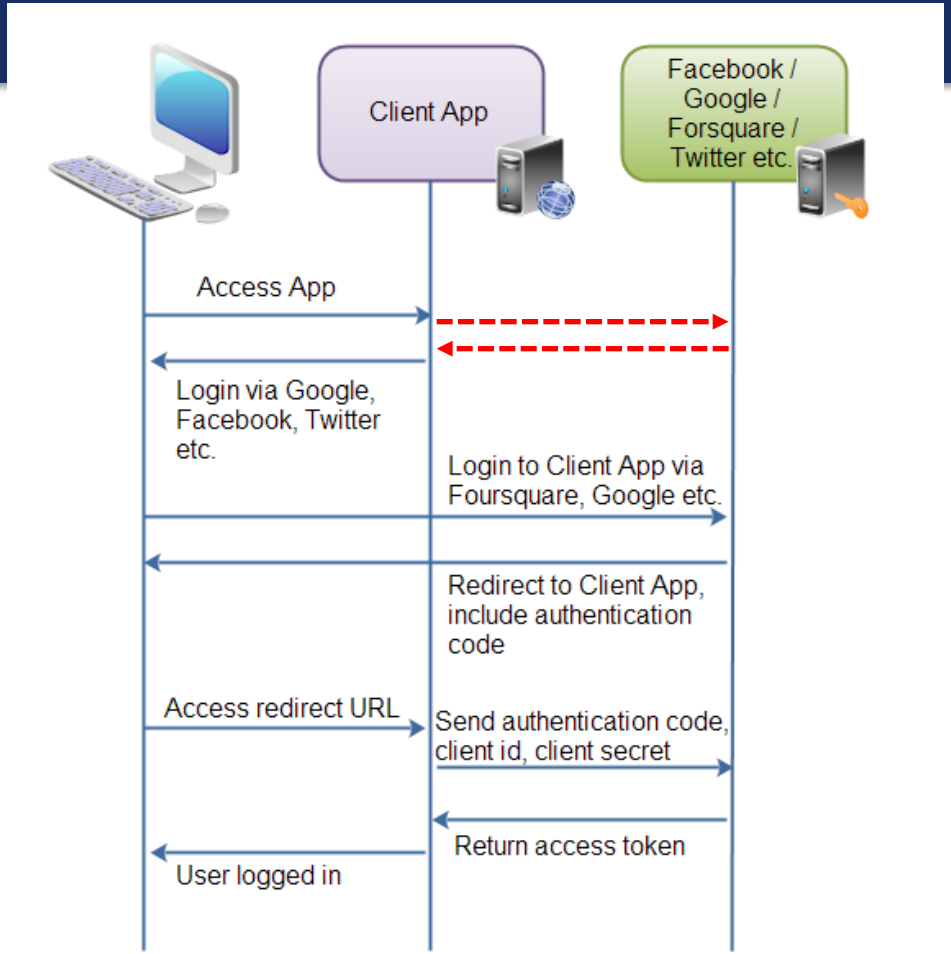
- Resource Owner
- Resource Server
- Client Application
- Authorization Server

These roles are illustrated in this diagram:

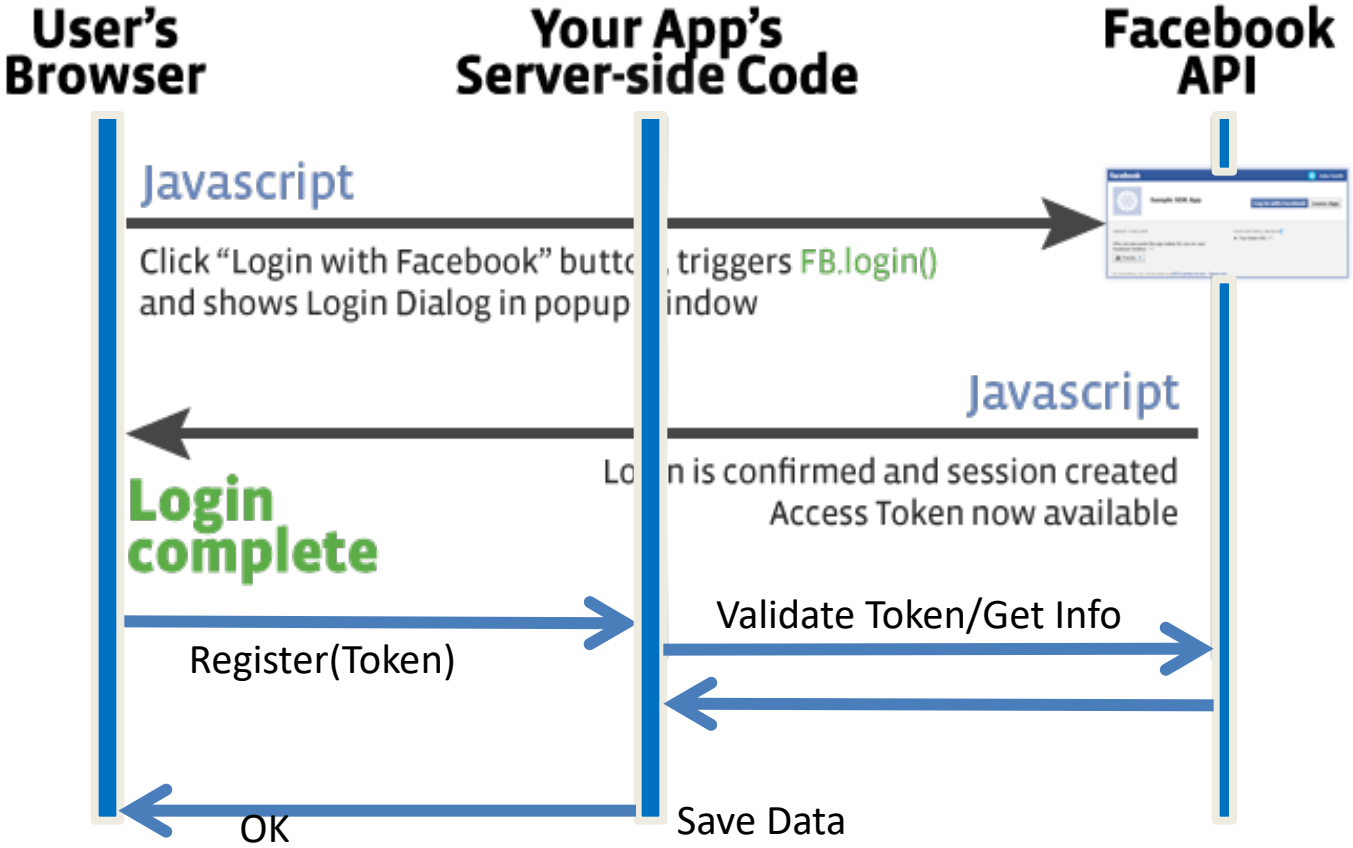


Roles/Flows

- User Clicks “Logon with XXX”
  - Redirect user to XXX
  - With Client App application ID.
  - And permissions app requests.
  - Code MAY have to call Resource Server to get a token to include in redirect URL.
- Browser redirected to XXX
  - Logon on to XXX prompt.
  - Followed by “do you grant permission to …”
  - Clicking button drives a redirect back to Client App. URL contains a temporary token.
- User/Browser
  - Redirected to Client App URL with token.
  - Client App calls XXX API
  - Obtains access token.
  - Returns to User.
- Client App can use access token on API calls.

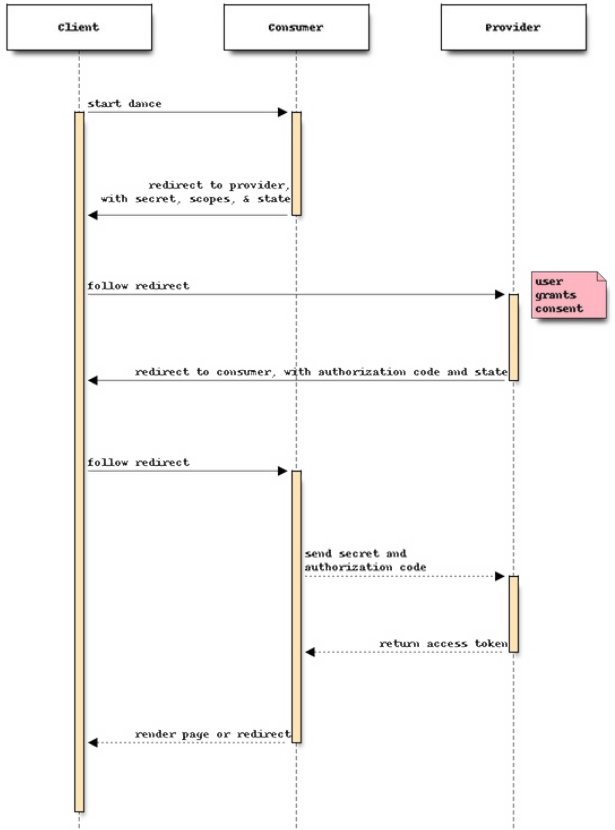


Facebook



Walkthrough: Middleware and Security

- Explain why I have to go incognito.
- Show:
  - Project code ... ...
  - GCP application registration page.
  - Token in the header (Postman)
- I normally do these things explicitly in my code.
  - I used Facebook in the past.
  - I decided to use Google because I expect people using my team management application to have Lionmail/Google accounts.
  - For expediency, I used Flask-Dance (https://flask-dance.readthedocs.io/en/latest/)
  - I understand the magic in general, but ...
  - I have not yet "learned the flask-dance magic" and how it uses the protocol under the hood.
- The net is:
  - OAuth2 is about *authorization,* but ...
  - Getting the basic profile proves *authentication*
  - And, I can use to simplify the registration process.



To Show

- Code walkthrough.
- Cookie/header.
- Certificate.
- … …

Event Drive Processing
Publish/Subscribe

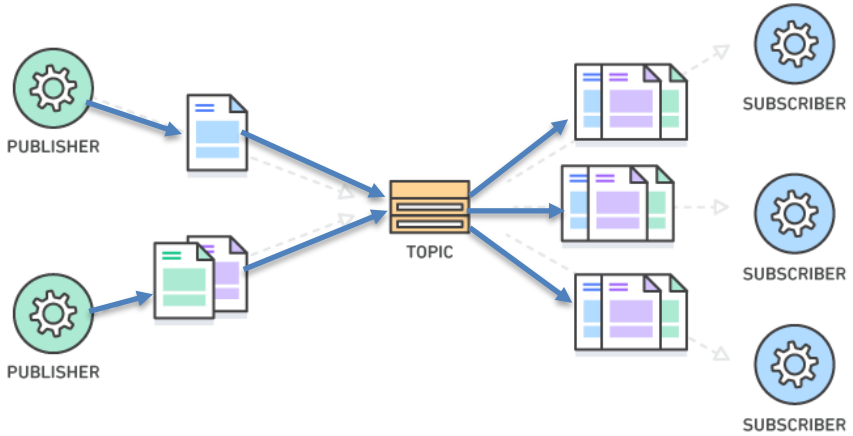
Pub/Sub – Event Driven Processing

"In software architecture, **publish–subscribe** is a messaging pattern where senders of messages do not program the messages to be sent directly to specific receivers, called subscribers, but instead categorize pub classes without knowledge of which subscribers, if any, there may be. Similarly, subscribers express interest in o and only receive messages that are of interest, without knowledge of which publishers, if any, there are.

Publish–subscribe is a sibling of the message queue paradigm, and is typically one part of a larger message-oriented middleware system. Most messaging systems support both the pub/sub and message queue models in their API, e.g. Java Message Service (JMS).

This pattern provides greater network scalability and a more dynamic network topology, with a resulting decreased flexibility to modify the publisher and the structure of the published data."

https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern

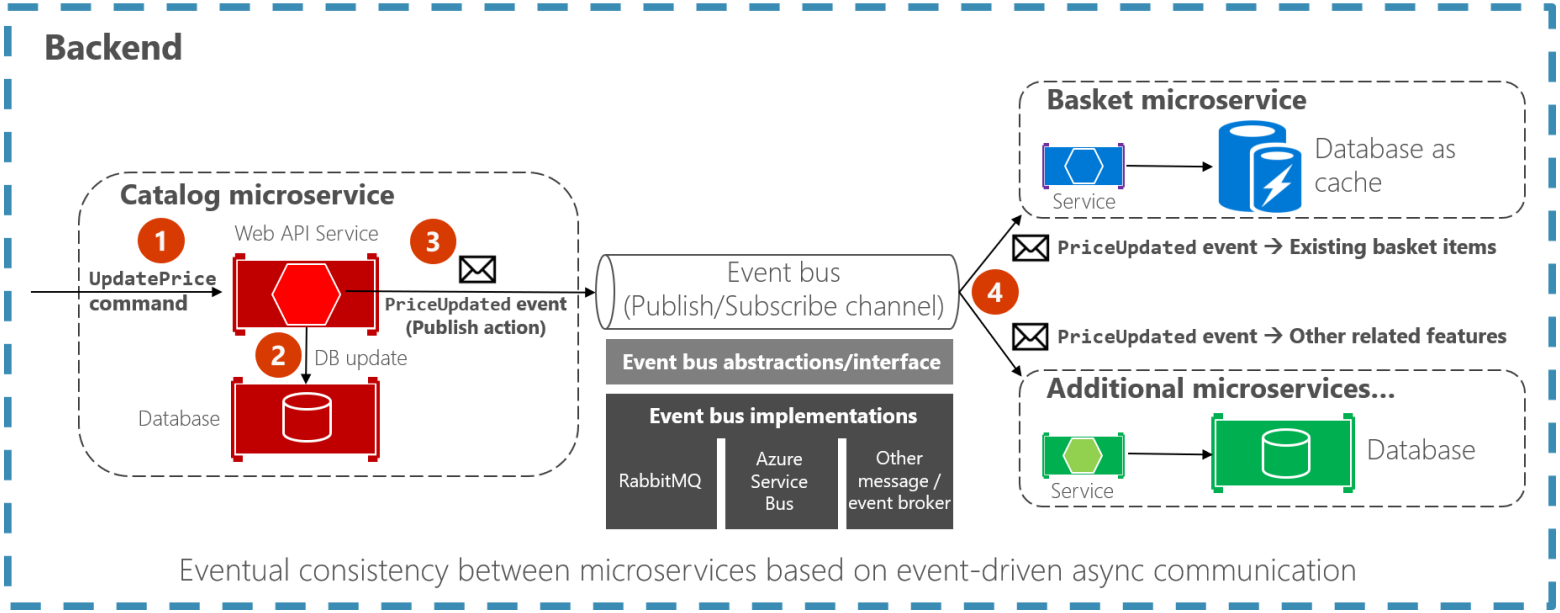


For microservices:

- It support loose coupling for application evolution and agility.
- I can add and evolve services without modifying the the base services.
- For example,
  - If creating a customer service **calls** "send an email."
  - I have to modify the original code to add support for SMS, webhook, etc.
  - With EDA, I just add or modify subscriptions.

Concept

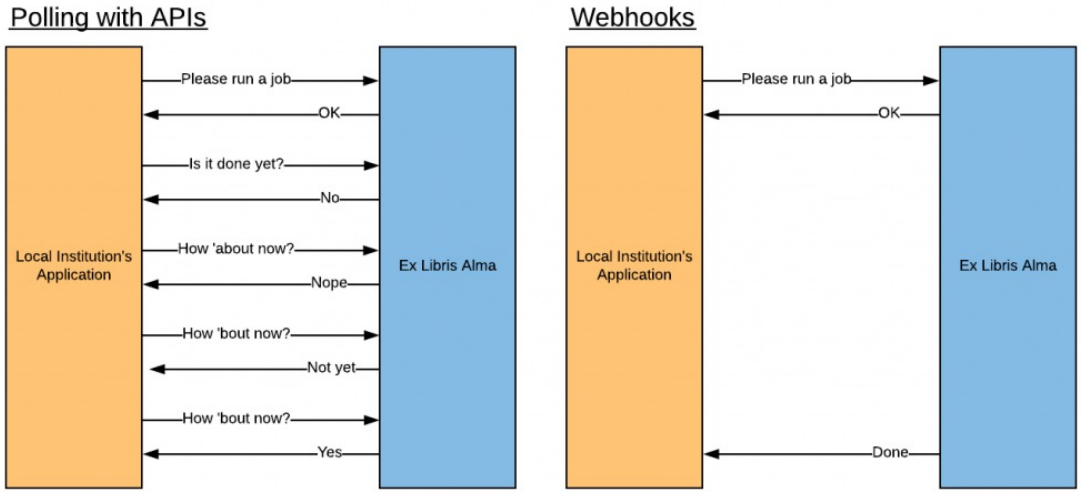
Implementing asynchronous event-driven communication with an event bus



https://docs.microsoft.com/en-us/dotnet/architecture/microservices/multi-container-microservice-net-applications/integration-event-based-microservice-communications

Webhooks

- “A webhook in web development is a method of augmenting or altering the behavior of a web page or web application with custom callbacks. These callbacks may be maintained, modified, and managed by third-party users and developers who may not necessarily be affiliated with the originating website or application. The term "webhook" was coined by Jeff Lindsay in 2007 from the computer programming term hook.”



Function-as-a-Service

Serverless and Function-as-a-Service

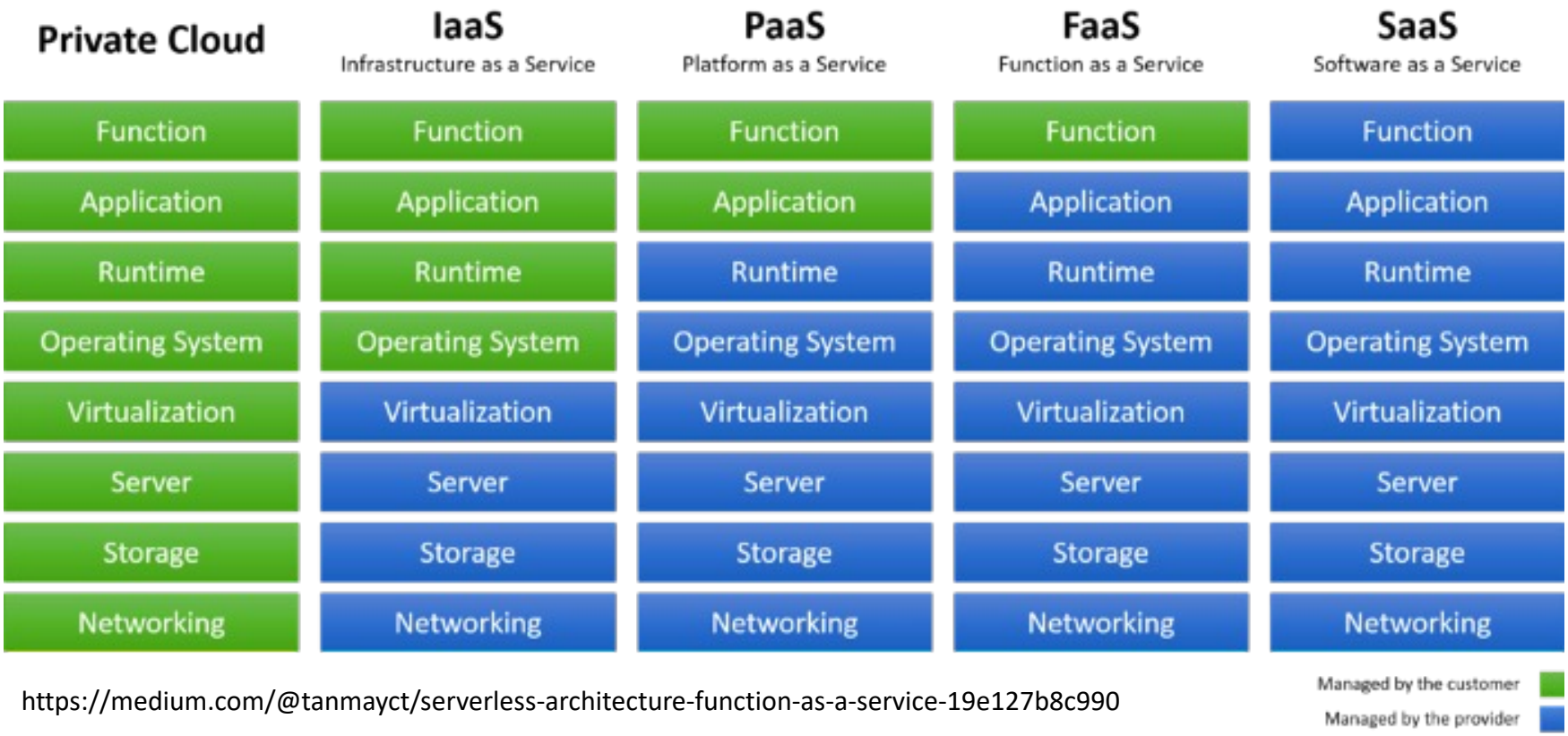
- **Serverless computing** is a cloud computing execution model in which the cloud provider runs the server, and dynamically manages the allocation of machine resources. Pricing is based on the actual amount of resources consumed by an application, rather than on pre-purchased units of capacity.[1] It can be a form of utility computing.

  Serverless computing can simplify the process of deploying code into production. Scaling, capacity planning and maintenance operations may be hidden from the developer or operator. Serverless code can be used in conjunction with code deployed in traditional styles, such as microservices. Alternatively, applications can be written to be purely serverless and use no provisioned servers at all.[2] This should not be confused with computing or networking models that do not require an actual server to function, such as peer-to-peer (P2P)."

- **Function as a service** (**FaaS**) is a category of cloud computing services that provides a platform allowing customers to develop, run, and manage application functionalities without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app.[1] Building an application following this model is one way of achieving a "serverless" architecture, and is typically used when building microservices applications."

- That was baffling:
  - IaaS, CaaS – You control the SW stack and the cloud provides (virtual) HW.
  - PaaS – The cloud hides the lower layer SW and provides an application container (e.g. Flask) with "Your code goes here." You are aware of container.
  - FaaS – The cloud provides the container, and you implement functions (corresponding to routes in REST).

Serverless and Function-as-a-Service



https://medium.com/@tanmayct/serverless-architecture-function-as-a-service-19e127b8c990

What is Serverless Good/Not Good For ... ...

Serverless is **good** for

short-running
stateless
event-driven

- Microservices
- Mobile Backends
- Bots, ML Inferencing
- IoT
- Modest Stream Processing
- Service integration

Serverless is **not good** for

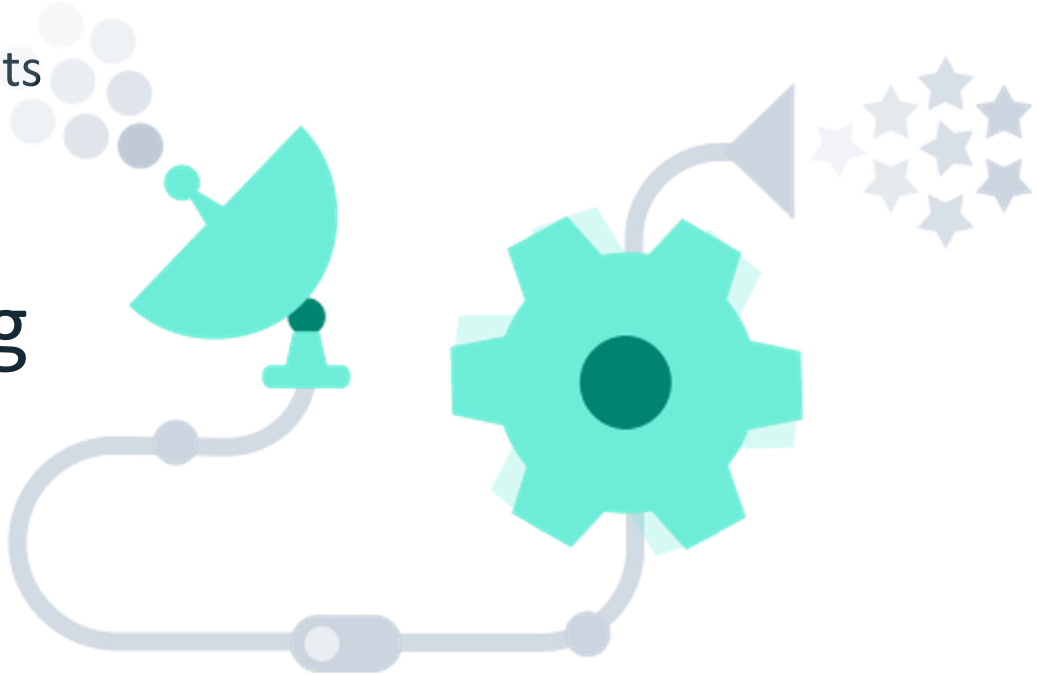
long-running
stateful
number crunching

- Databases
- Deep Learning Training
- Heavy-Duty Stream Analytics
- Numerical Simulation
- Video Streaming

What triggers code execution?

Runs code **in response** to events

Event-programming model



(Some) Current Platforms for Serverless

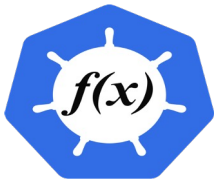
AWS Lambda

OpenLambda

Azure Functions

Kubernetes

IBM Cloud Functions



Red-Hat

Google Functions

Let’s Build a Lambda Function

- We are going to do it manually to understand the process/steps.

- There are several more complete approaches:
  - Pipelines
  - Serverless Framework (https://www.serverless.com/framework/docs/providers/aws/guide/intro)
  - AWS Toolkit for PyCharm (https://aws.amazon.com/pycharm/)

- We will explore some of these … …

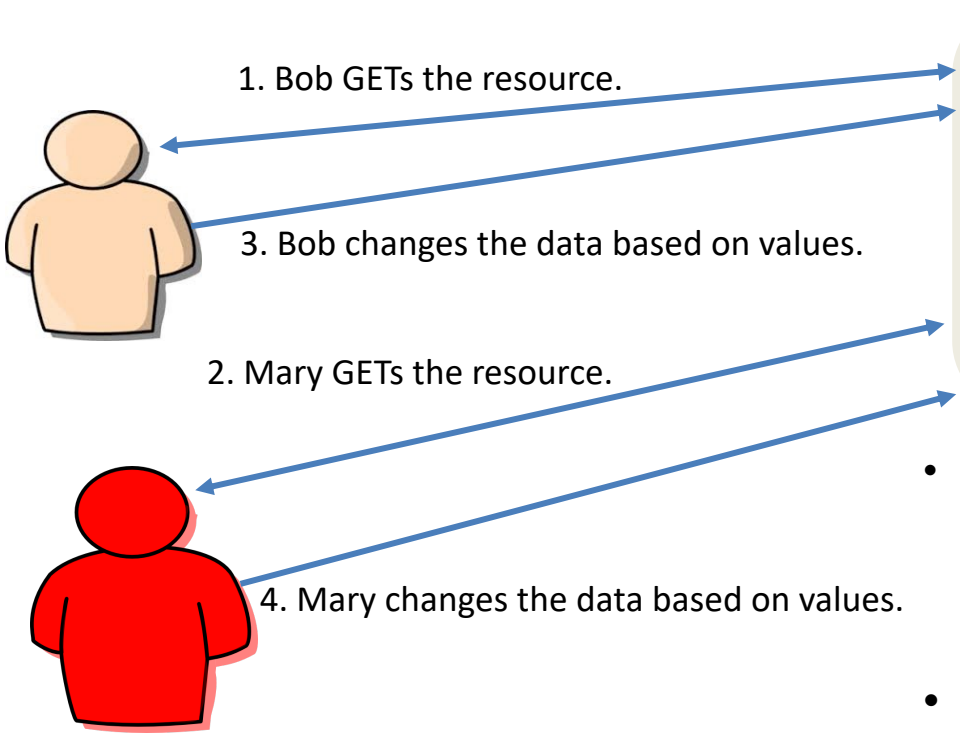
Putting Some Pieces Together

Projects

- /Users/donaldferguson/Dropbox/00NewProjects/SimpleMiddlerware
- /Users/donaldferguson/Dropbox/00NewProjects/slack-lambda/hello_world/slack_connector.py
- E6156-Student-Event Lambda function via Lambda Console
- /Users/donaldferguson/Dropbox/00NewProjects/google_login

ETag, Conditional Update

REST and Isolation: Read then Update Conflict



- If
  - The written value
  - Depends on the read value.
  - The resource may be in the wrong state at the end.
- The execution was RRWW
- But the intent was RWRW.

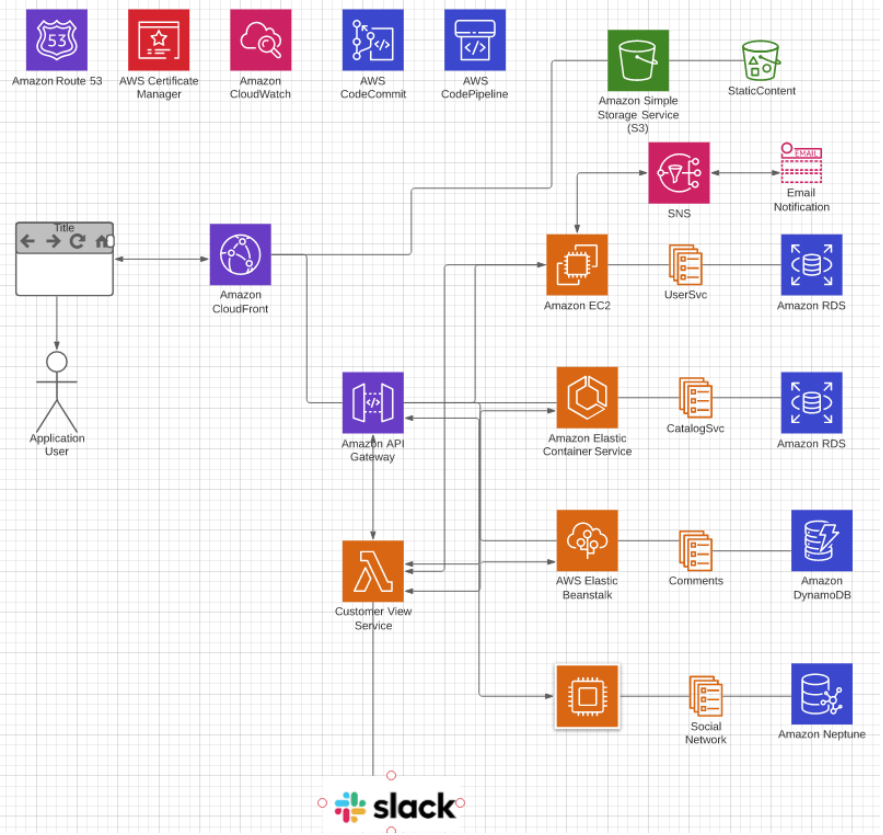
Isolation/Concurrency Control

This requires conversation state, which would violate the REST Stateless principle.

- There are two basic approaches to implementing isolation
  - Locking/Pessimistic, e.g. cursor isolation
  - Optimistic: Before committing, each transaction verifies that no other transaction has modified the data it has read.
- How does this work in REST?
  - The server maintains an ETag (Entity Tag) for each resource.
  - Every time a resource's state changes, the server computes a new ETag.
  - The server includes the ETag in the header when returning data to the client.
  - The server may *optionally* maintain and return "Last Modified" time for resources.
- Semantics on updates
  - If-Match – value of a previous calls ETag response used with a PUT or DELETE. Server should only act if nobody else has modified the resource since you last fetched it. Otherwise provides a 412.
  - If-Modified-Since – value of a previous Last-Modified response used with a GET. Server should only provide a response if the resource was modified since the timestamp submitted. Use in conjunction with If-None-Match in case the change occurred within the same second. Otherwise provide a 304.
  - If-None-Match – value of a previous calls ETag response used with a GET. Server should only provide a response if the ETag doesn't match, i.e. the resource has been altered. Otherwise provide a 304.
  - If-Unmodified-Since – value of a previous Last-Modified response used with a PUT or DELETE. Server should only act if nobody else has modified the resource since you last fetched it. Otherwise provides a 412

Hands-On CloudFront
DNS, Certificate, HTTPS

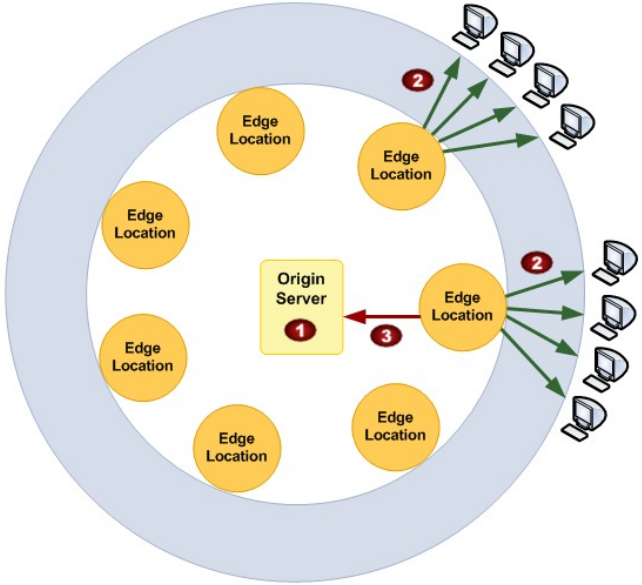
Overall Architecture

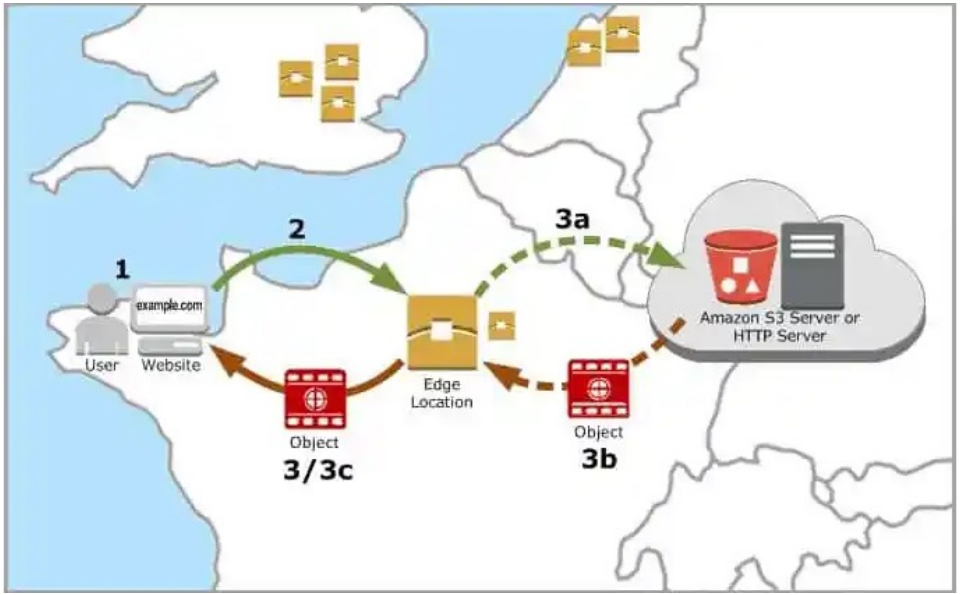


- There may be multiple
  - Microservices
  - Web UI projects/content
- CloudFront
  - Multiple S3 content → one site.
  - Forward API requests to API GW
- API GW manages/monitors APIs
- Networking/HTTPS requires
  - Domain, DNS
  - Certificates

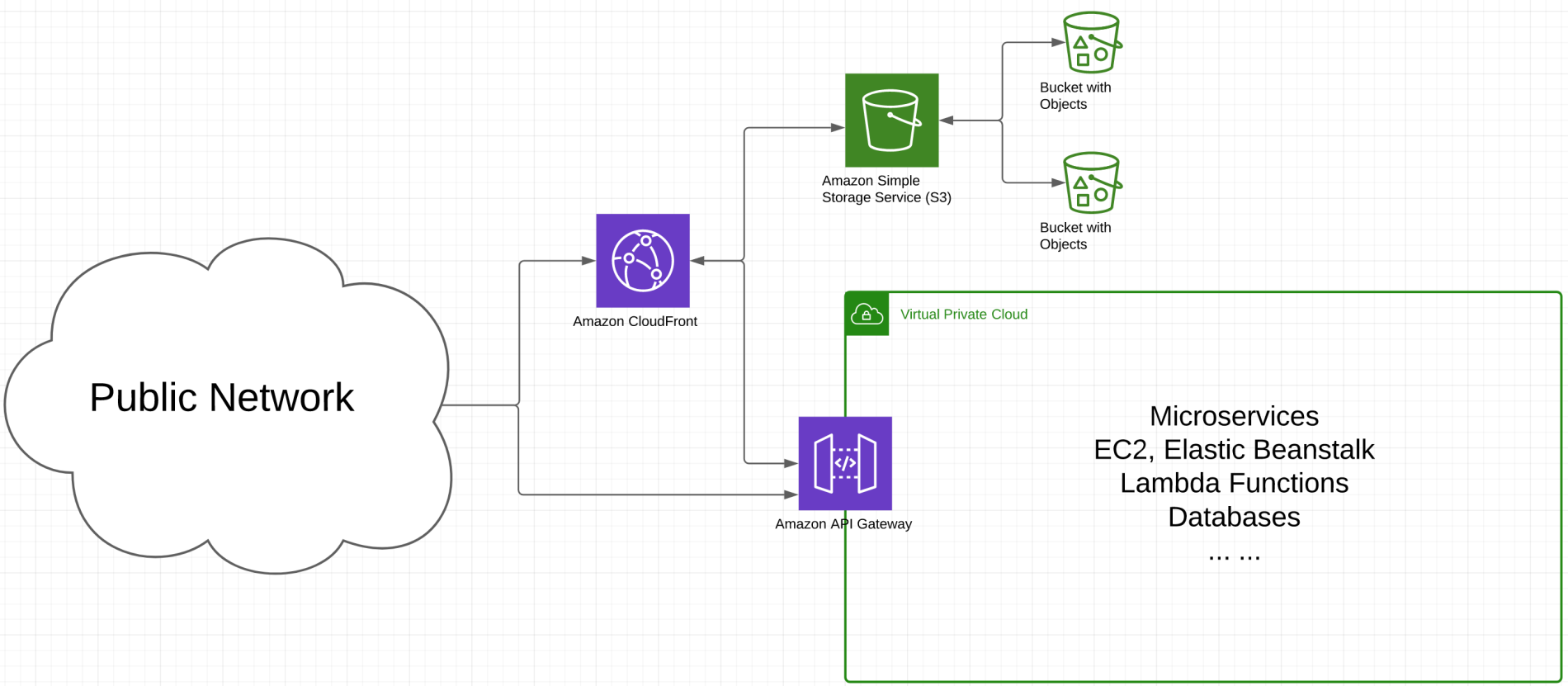
CloudFront

"A content delivery network, or content distribution network (CDN), is a geographically distributed network of proxy servers and their data centers. The goal is to provide high availability and performance by distributing the service spatially relative to end users." (https://en.wikipedia.org/wiki/Content_delivery_network)

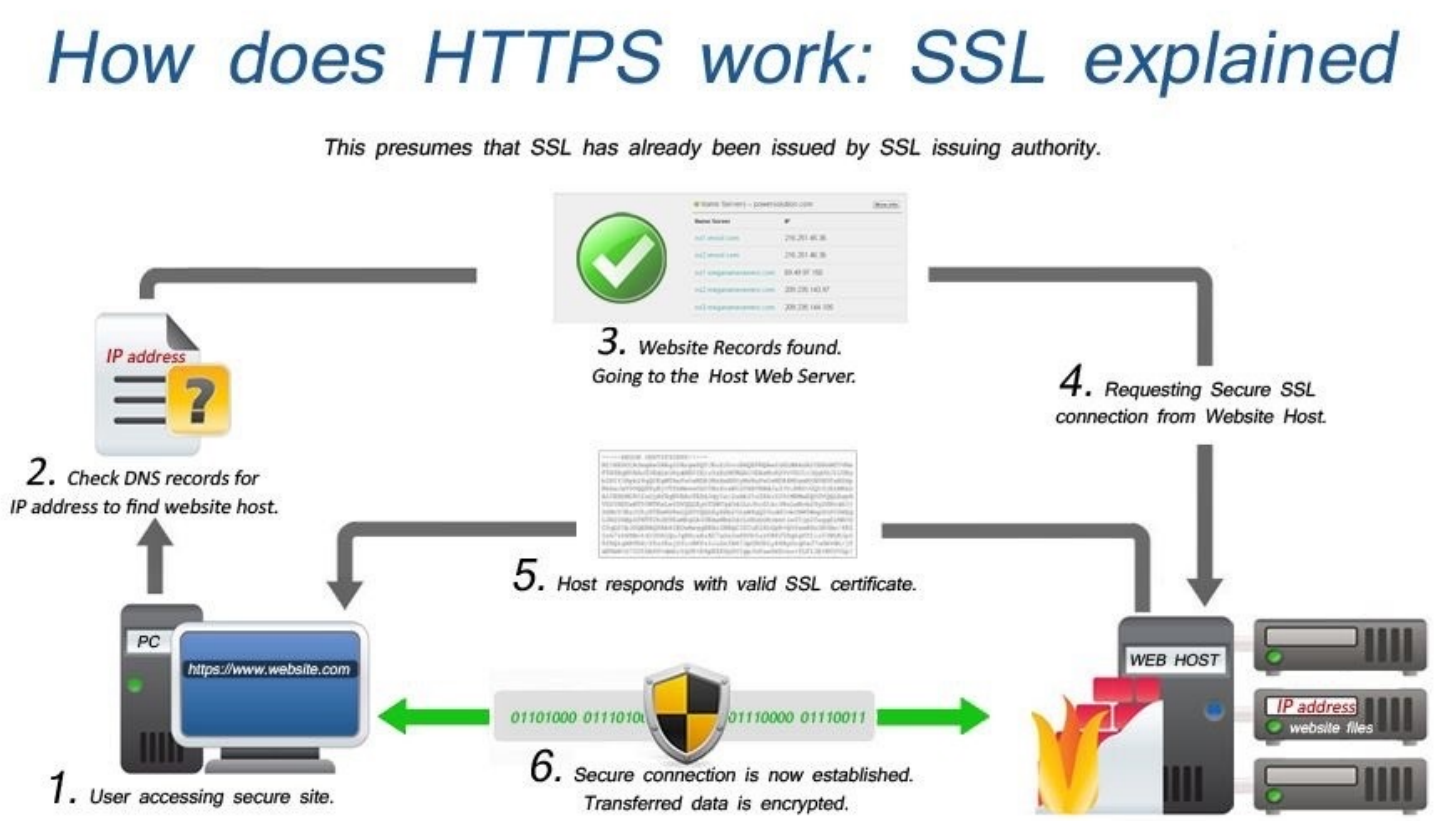




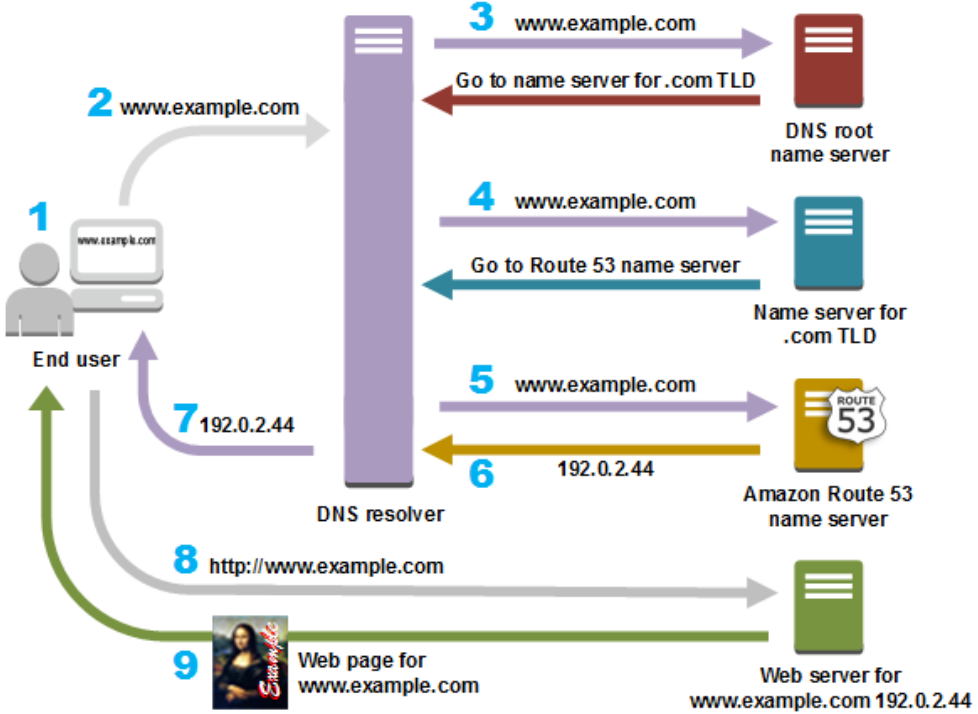
Big Picture



Certificates and HTTPS



DNS



Things to Demo

- CloudFront:
  - Distributions, Origins, Behavior
  - Monitoring, caching, alarms, ...
- Certificates
  - AWS implementation
  - Use in CloudFront and elsewhere
  - Visibility in browser, HTTPs
- Route 53
  - Proving ownership
  - Records